Report of Natural Language Processing, Final Project

Group 4

Evidence Sentence Extraction

We use TF-IDF (Term Frequency Inverse Document Frequency of records) for evidence sentence extraction and implemented this by existing sklearn package:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

First, fit "all the sentences" to the vectorizer. Then, we transform each "preprocessed article sentence" by the vectorizer to get the evidence sentences.

```
vectorizer = TfidfVectorizer()
vectorizer.fit(all_rel_sentences)
for sentence in article:
    tfidf_vectorizer.transform(article_sentences)
```

Since vectorizing the whole dataset is a time-and-resource-consuming task, we have only done it twice (with and without preprocessing the sentences). The result is that with sentences being preprocessed, we get a higher performance.

Our preprocessing methods:

```
def preprocess_sentence(sentence):
    sentence = sentence.lower()
    # remove link, website or short keywords
    if ('link:' in sentence) or (r'http' in sentence) or len(sentence.split()) <= 4:
        return ""
    # remove punctuation
    sentence = sentence.translate(str.maketrans('', '', string.punctuation))
    # remove additional space
    sentence = sentence.replace(r'\s+', ' ')
    # print(sentence)
    return sentence
```

Comparison

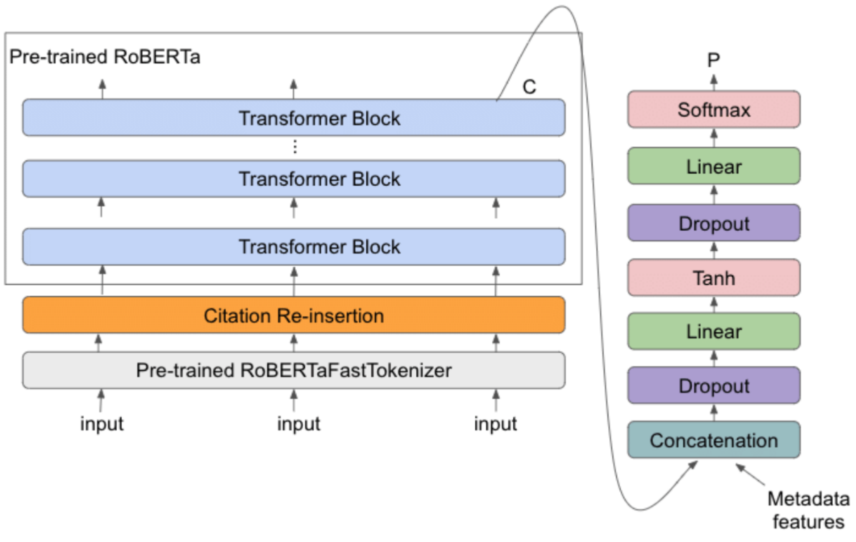
	preprocess	without preprocess
public score on kaggle	0.56584	0.55944

Sequence Model

We use transformers to build a RoBERTa model for sequence classification. RoBERTa model is a transformer-based language model that uses self-attention to process input sequences and generate contextualized representations of words in a sentence.

- **Model Architectures**

  This implementation is the same as Bert model with a tiny embeddings tweak as well as a setup for Roberta pre-trained models; however, it uses a byte-level BPE as a tokenizer (`RobertaTokenizer`) and uses a different pre-training scheme.



- **Hyperparameters**
  - evaluation_strategy = "epoch"
  - save_strategy = 'epoch'
  - learning_rate = 5e-6
  - per_device_train_batch_size = 24
  - per_device_eval_batch_size = 24
  - num_train_epochs = 3
  - weight_decay = 0.001
  - load_best_model_at_end = True
  - metric_for_best_model = 'macro f1-score'
  - seed = 42

- **Loss Function**

  Computed by Trainer(), loss is returned with outputs[“loss”] when label is provided. RoBERTa model computed the Masked language modeling (MLM) loss.

```
def compute_loss(self, model, inputs, return_outputs=False):

        if self.label_smoother is not None and "labels" in inputs:
            labels = inputs.pop("labels")
        else:
            labels = None
        outputs = model(**inputs)

        if self.args.past_index >= 0:
            self._past = outputs[self.args.past_index]

        if labels is not None:
            loss = self.label_smoother(outputs, labels)
        else:
            loss = outputs["loss"] if isinstance(outputs, dict) else outputs[0]

        return (loss, outputs) if return_outputs else loss
```

- **Input**
  - input_ids: indices of input sequence tokens in the vocabulary
  - attention_mask: mask to avoid performing attention on padding token indices (values selected in [0, 1])
  - token_type_ids: segment token indices to indicate first and second portions of the inputs (indices are selected in [0, 1])
  - labels: labels for computing the sequence classification loss (indices should be in [0, 1, 2])

- **Output**
  - loss
  - logits: prediction scores

    (for each vocabulary token before SoftMax function)

Comparison

We compare different models, learning rate, and save_strategy below.

model candidate

1. ‘roberta-base’, using the BERT-base architecture, has 12-layer, 768-hidden, 12-heads, 125M parameters.
2. ‘roberta-large’, using the BERT-large architecture, has 24-layer, 1024-hidden, 16-heads, 355M parameters.

	roberta-base	roberta-large
public score on kaggle	0.56584	0.4687

learning rate

(epochs = 3, evaluation_strategy = "epoch", save_strategy = “epoch”)

	1e-5	5e-6	1e-6	5e-7
public score on kaggle	0.45233	0.56584	0.55614	0.41526

save_strategy

Parameter save_strategy is the checkpoint to adopt to save strategy during training.
possible values are:

- **epoch:** save is done at the end of each epoch
- **steps:** save is done every save_steps
- **no:** no save is done during training

(learning rate = 5e-6, epochs = 3)

	epoch	step	no
public score on kaggle	0.56584	0.40968	0.33039

Error Analysis

We use classification_report from sklearn to analyze our model.

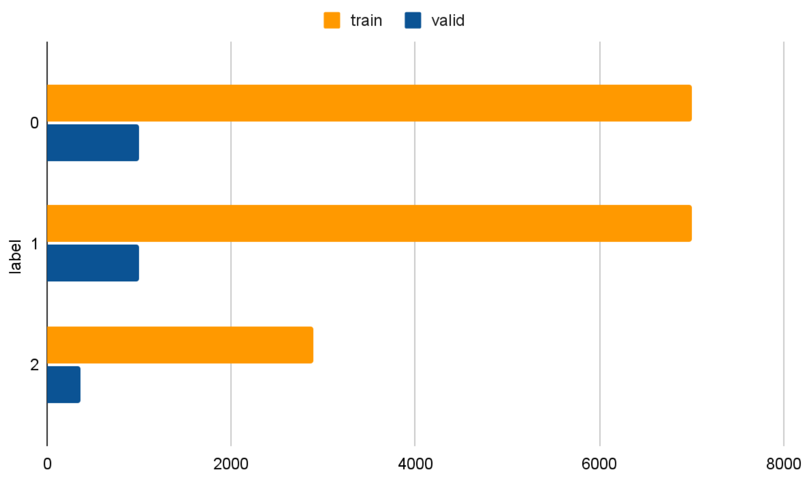
```python
from sklearn.metrics import accuracy_score, classification_report
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    accuracy = accuracy_score(labels, predictions)
    target_names = [0, 1, 2]
    output_dict = classification_report(labels, predictions, labels=range(3),
                                        target_names=target_names, output_dict=True)
    return {'accuracy': accuracy, 'macro f1-score': output_dict['macro avg']['f1-score']}
```

Use pretrain_model = 'roberta-large' and train three epochs for example:

Epoch	Training Loss	Validation Loss	Accuracy	Macro f1-score
1	0.985900	0.924528	0.573611	0.414463
2	0.913000	0.898621	0.586339	0.443739
3	0.884000	0.891291	0.592703	0.470188

Unbalanced Data Label

One time, we got an error from classification_report that said “Number of classes, 2, does not match size of target_name, 3”. We then perceived that due to the unbalanced number of labels, the output barely gives the prediction of label 2. Our solution is to add parameter labels=range(3) inside classification_report to avoid this kind of error.



Quality Inconsistency

After our preprocess function, there are *three* claims that result in “null article”. We went through the dataset and found that the .json files only contain some “links”, i.e. youtube links, rather than “strings”. And the link is wiped out through our preprocess function. For the three claims mentioned above, we simply key in label “0” at the end of prediction.

```
["Link: canonical", "Link: alternate", "Link: alternate",
"Link: shortlinkUrl", "Link: alternate", "Link: image_src",
"Link: https://www.youtube.com/watch?v=s5_XYrtSMRs",
"Link: http://www.youtube.com/user/BillWhiteForTexas",
"Link: https://i.ytimg.com/vi/s5_XYrtSMRs/hqdefault.jpg",
"Link: https://i.ytimg.com/vi/s5_XYrtSMRs/hqdefault.jpg",
"Link: https://www.youtube.com/embed/s5_XYrtSMRs", "_____________________"]
```

Above is one example of premise_article containing links rather than strings

Reproduce the prediction from preprocessed data

- Download the preprocess data here and put them into the folder “data”. (original data is unnecessary)
- Download the model checkpoint here and put them into the folder “checkpoint-5620”
- Run inference.ipynb and get “output/prediction.csv”.

Reproduce the prediction from original data

- Put the dataset into ./data folder or modify location of `dataset_path = "./data"` in Line 1 of the last cell in preprocess.ipynb.
- Run preprocess.ipynb
- Follow the procedures above.